Static Incremental Font Transfer

Garret Rieger - Nov 2023

Limitations of Current IFTB Proposal

- Designed to only augment outline data. For many fonts this makes sense but for some types of families non-outline data can be significant. (eg. complex GSUB/GPOS, COLRv1)
- Challenges in encoding fonts that have lots of complex interactions between glyphs.
  - Solved by either duplicating or including glyphs in the base file, which can be inefficient.
- Variable design space augmentation is not supported.

Solution

- Taking inspiration from patch subset, we can use generic binary diffs via shared brotli patches.
- This allows all data within the font to be augmented.
- Expands the utility of IFTB, would be an optional additional patch type.
- Unlike patch subset, the patches wouldn’t be requested via a dynamic protocol. They would just be regular statically hosted files.

Goals

- Extend the IFTB proposal to be more general purpose by allowing the optional use of shared brotli patches.
- Eliminate the need for the separate patch subset protocol.
- Unlike patch subset can be implemented entirely statically, with the option for a dynamic implementation if desired.
- Experiment with alternate encodings of the IFTB table in order to support the optional use of generic binary diffs.

The Model

- A subset definition is a set of codepoints, layout features, and design space to be added to a font.

- Within the font file there is a map embedded which maps from:
  Subset definition -> patch URL

- Patches can have different encodings, to start we'll have:
  - IFTB Chunk
  - Shared Brotli
  - But more could be added later.

The new Model

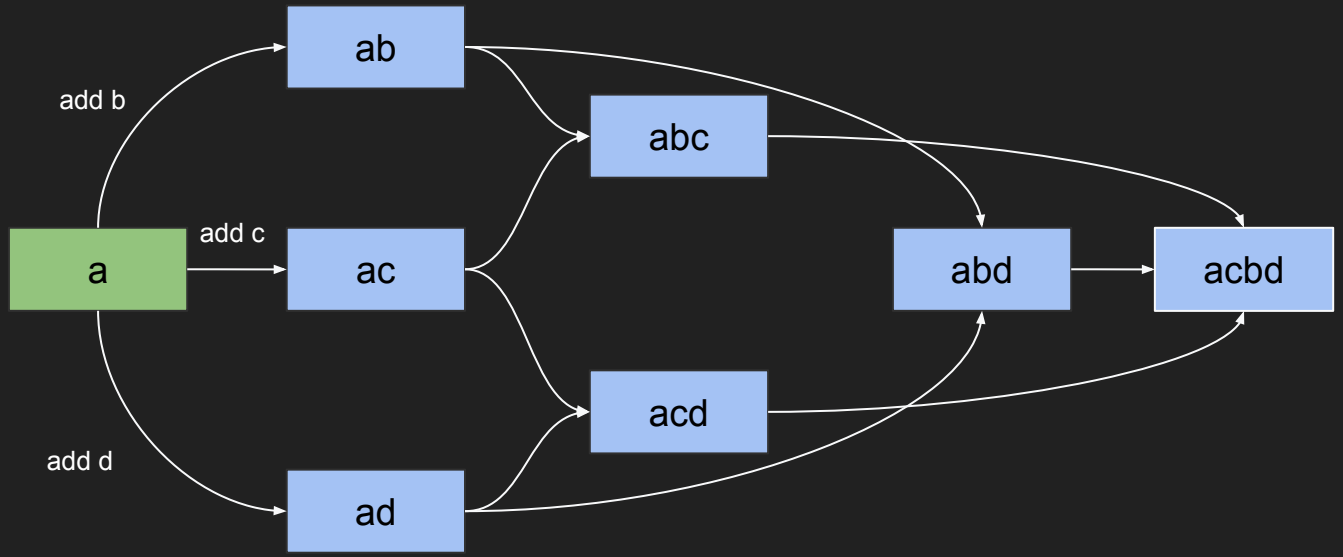
- A patch encoding is a function which takes an input font and patch data and outputs an extended version of the font.

- Encodings can be one of two types:
  - **Independent:** patches are commutative. (eg. IFTB)
  - **Dependent:** patches are not commutative. (eg. shared brotli).

- Since the mapping is embedded in the font file patches can modify it.

- With dependent patches this allows the construction of a graph.

Patch Graph for 4 subsets: a, b, c, and d



Each edge in the graph is a patch

Extensions to IFTB

Some changes to the current IFTB proposal will be needed:

1. Map from (codepoints, layout features, design space) to patch URL.
   IFTB currently maps from (gids, layout features) to patch URL.
   a. Features + design space are optional.

2. Optionally allow opaque string ids instead of numeric chunk indices when identifying patch URLs.
   a. Helpful for dynamic implementations.

3. Associate an encoding type with each patch URL.

Prototype IFT table Encoding

Header

- URL template.
- File ID
- Default Patch Encoding

Patch Map

Subset Definition
- Bias (int)
- Biased Codepoints (sparse bit set)
- Feature tag set * (tag set encoding)
- Design space. *
- Previous subsets definitions indices.

- patch id (int)
- patch encoding (int)

Patch IDs (Optional)
- Patch id * (int)

- Id token * (string)

* unimplemented in current prototype.

Compression Techniques

Keeping the encoded size (post brotli compression) of the mapping table small is important:

- Codepoint sets use the sparse bit set encoding developed for patch subset:
  - A bias value improves encoding cost when codepoints in the set are close together.
- Patch indices are encoded as deltas with an implicit +1 from the previous entry:
  - Allows them to be completely omitted in most cases.

Compression Techniques (continued)

- Only specify non default encodings.
  - Default encoding declared in header, only need to encode when different from default.
- Allow subsets to be formed by referencing previous subsets.
  - Can compactly represent combinations or extensions of previously defined subsets. (eg. subset X + an additional layout feature).

Patch Encodings

IFTB Patches:

- Encoding for IFTB chunks is unchanged from the current proposal.

Shared Brotli Patches:

- Just a raw shared brotli patch, but might consider adding a short header.

Demo

https://garretrieger.github.io/ift-demo/

Demo

https://garretrieger.github.io/ift-demo/

Fully static demo:

- Augments Roboto for (basic latin, vietnamese, and cyrillic) w/ shared brotli.
- Augments Noto Sans/Serif SC w/ IFTB patches.
- All fonts using new prototype mapping table.

Post Compression Encoding Costs

Font	Chunk Size (kb)	IFT Total Base Size (woff2, kb)	IFTB Total Base Size (woff2, kb)	Delta (kb)
Noto Sans/Serif SC (TTF)	110	398	393	+5
Noto Sans/Serif SC (TTF)	49	407	402	+5
Noto Sans/Serif SC (TTF)	4.1	450	480	-30
Noto Sans/Serif SC (TTF) (Hi Freq Only)	4.1	84	83	+1
Noto Sans/Serif SC (TTF) (Low Freq Only)	4.1	169	174	+5

Demo Total Transfer Sizes

CJK Presplit?	Chunk Size (kb)	IFT Transferred (kb)	Patch Subset (kb)	Unicode Range (kb)
No	110	2,478	165	1,786
No	49	2,058	165	1,786
No	4.1	1,062	165	1,786
Yes	4.1	706	165	1,786

Prototype Implemented So Far:

- Conversion utilities (iftb to ift, ttf to ift).
- Simple encoder library for shared brotli graphs.
- For IFTB patches, re-uses Skef's IFTB library.
- Client library.
- IFT table parsing and compiling (using protobufs).
- JS Client via WASM.
- Code available here: https://github.com/w3c/patch-subset-incxfer
- Writeup: https://github.com/w3c/patch-subset-incxfer/blob/main/docs/static-ift.md

Prototype not Implemented Yet:

- IFTB CFF Support (just missing char strings offset, should be easy to add).
- Feature tags and design space.
- Overlapping Subset Definitions.
- Optional patch id to id string mapping.
- Table encoded via protobuf, eventually want a custom binary encoding based on the existing IFTB table proposal.

What’s Next

- I’ve got additional ideas to further reduce encoding cost.
- Experiment with alternate encoding techniques.
- Implement missing prototype features (particular layout tags, design space, and combined subsets).
- Experiment w/ per table shared brotli patching. This might enable mixing shared brotli and IFTB patches in the same family for some use cases.

The IFTB GID Mapping Table

- For augmentation cases where glyph ids are stable and the full cmap is present in the base font, the IFTB gid mapping table works really well.
- It’s compact w/ brotli compression, and is very fast to interact with in the client.
- In many other font tables it’s common to have multiple sub formats that work better in specific cases (eg. cmap, GSUB, GPOS).
- I suggest we have both the general purpose encoding and the IFTB gid mapping encoding available as sub formats within the mapping table.

Naming

- Suggest we drop “Binned” from “Binned Incremental Font Transfer” and just call this “Incremental Font Transfer” since it becomes the one single method.